Automatically Evolving Lookup Tables for Function Approximation

Oliver Krauss
Johannes Keppler University Linz
University of Applied Sciences Upper Austria
oliver.krauss@fh-hagenberg.at

William B. Langdon
University College London
W.Langdon@cs.ucl.ac.uk

Abstract

- Big picture: Genetic Improvement to *automatically create or update constants* in software
- This publication: *creation of lookup tables* for methods that can be *approximated*
- Contribution: *high precision* and *fast run-time* performance.
- Results: *double precision* accuracy outperforming reference implementations
- Limitations: *function range* and functions with *inflection points*

Introduction

Newton-Raphson approximation with 3 iterations to improve *run-time and accuracy* of math functions. Research is about *automatically generating lookup tables* for x_0.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (1)$$

Contributions:

- Double precision accuracy (1 sign bit, 11 bit exponent, 52 bit fractional [1])
- Automated generation via user provided function
- Fast run-time performance compared to not hardware-accelerated functions

Background

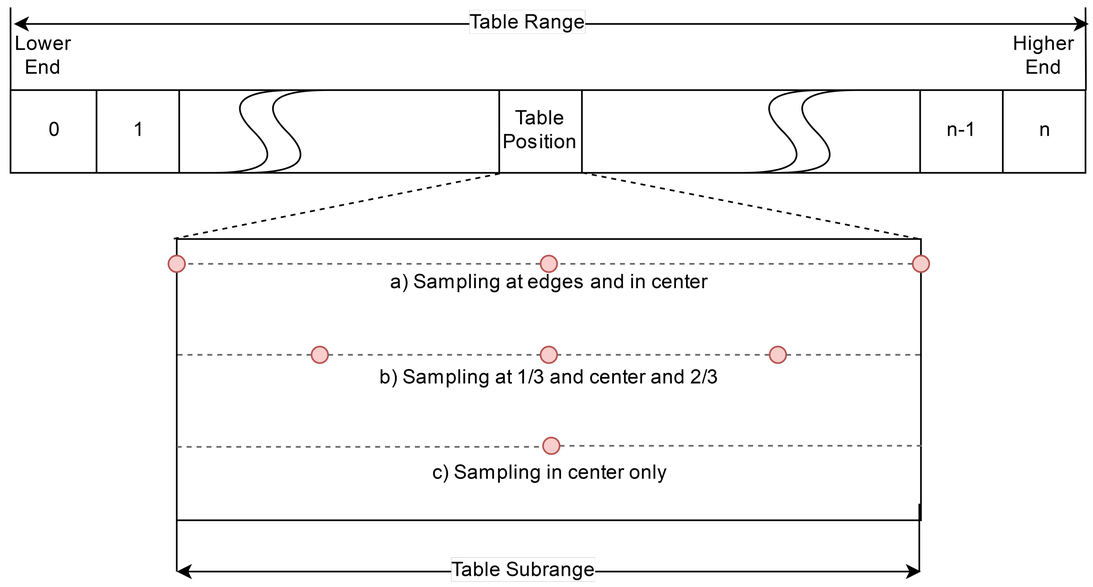
This work utilizes Covariance Matrix Adaptation-Evolution Strategy (CMA-ES), and is based on Evolving better Software Parameters [2]. They adapted *glibc square root* to *cube root* with a lookup table *automatically generated by CMA-ES*, and outperform Java and C++ in accuracy).

Implementation	Distribution	Total Error $\times\ 10^{-10}$
C - with log [2]	even	3.1451
	random	3.3231
Java	even	3.3322
	random	3.6071
C++	even	6.2851
	random	7.2275

Methods

The approach from [2] was generalized to support *any given mathematical function*. The research is focused on finding a good *fitness function*. Using a combination of sampling points for the lookup table entries, adaptions to the quality measure close to the optimal 0 (multiplier, logarithm, etc.), and the following measures:

- Approximation - $fitness = abs(fn(newtonRaphson(input)) - input)$
- Remaining Error - $fitness = remainingErrorNewtonRaphson(input)$
- Direct - $fitness = abs(fn(tableEntry) - input)$



Results

Testing was conducted on all combinations of fitness functions (35) on square-, cube-, and super-root as well as a function with multiple inflections, and a function with a single inflection (inflection point outside of lookup table range)

Fitness Function

- In even distribution sampling at edges works best
- In random distribution sampling only center works best
- Applying Log (and other operations) to fitness function improves outcome
- On inflections, some fitness functions prevented convergence of CMA-ES

Comparison to Java, C, C++

- Slightly less accurate in Square Root
- Outperforms C++ and Java in Cube Root and Super Root

The run-time performance is competitive with software solutions, but not with hardware accelerated functions.

	Mean (in nanoseconds per call)		
Language	sqrt	cbrt	surt
Hardware Accelerated C	0.88	0.88	0.88*
Hardware Accelerated C++	4.10	21.35	8.10*
Langdon and Petkes cbrt		24.96	
Our approach	25.33	27.46	29.58
Java	1.02	69.51	1.03*

*surt implemented as sqrt(sqrt(x))

Conclusion and Outlook

- Use Case *functions without hardware acceleration*
- Achieving *double precision accuracy* and *fast run-time* with *smooth functions*
  - Limited to *specific range*
  - Does not work with *inflections*
  - Robustness of function has to be considered
- Big Picture: Fitness function design imperative for updating and generation of constants

References

[1] IEEE. Standard for Floating-Point Arithmetic. *Std 754-2008*, Aug 2008.

[2] William B. Langdon and Justyna Petke. Evolving Better Software Parameters. In *Search-Based Software Engineering*, pages 363–369. Springer, 2018.